# 1. vector (มาตรฐาน C++11)

vector เป็นหนึ่งใน C++ template class ในกลุ่มของ container จาก Standard Template Library (STL) และ vector เป็นส่วนหนึ่งของ STL ตั้งแต่มาตรฐานแรกของ C++ (C++98) เช่นเดียวกับ list, stack, queue, map, unorder_map, set และ priority_queue

vector เป็น container ที่เก็บข้อมูลทั้งหมดไว้ในหน่วยความจำที่มีตำแหน่งต่อเนื่องกันเช่นเดียวกับอาร์เรย์ในทางปฏิบัติแล้ว vector สร้างมาจากอาร์เรย์นั้นเอง เพียงแต่ vector เป็นอาร์เรย์แบบที่เรียกว่า dynamic array ซึ่งก็คืออาร์เรย์ที่ไม่จำเป็นต้องกำหนดขนาดเมื่อแรกสร้าง ขนาดของอาร์เรย์จะเพิ่มขึ้นและลดลงไปตามจำนวนข้อมูลที่อยู่ในอาร์เรย์อัตโนมัติ

ในด้านการพัฒนาแล้ว vector จะสร้างอาร์เรย์ที่กำหนดขนาดเริ่มต้นไว้ (capacity) หากเมื่อมีการบรรจุข้อมูลลงในอาร์เรย์จนเกินกว่าที่ขนาดของอาร์เรย์จะรองรับได้ จะมีการสร้างอาร์เรย์ที่ใหญ่กว่าเดิม (reallocation) จากนั้นทำการคัดลอกข้อมูลจากอาร์เรย์เดิมไปใส่ในอาร์เรย์ใหม่ รวมถึงนำข้อมูลใหม่ที่ไม่สามารถบรรจุในอาร์เรย์เดิมได้ ไปใส่ในอาร์เรย์ใหม่ วิธีการพัฒนาแบบนี้เป็นที่มาของจุดแข็งและจุดอ่อนของ vector โดยเฉพาะเมื่อเทียบกับ list (ที่พัฒนามาจาก linked list)

การเรียกใช้งาน vector ต้องใช้คำสั่ง preprocessor directive `#include<vector>`

**การประกาศ vector** ใช้รูปแบบต่อไปนี้

- `vector<type> vec` เพื่อสร้าง vector ที่ไม่มีข้อมูลใด ๆ (empty vector) ซึ่งมีขนาดเป็น 0 (Time Complexity $\theta(1)$)
- `vector<type> vec(N)` เพื่อสร้าง vector ขนาด `N` (Time Complexity: $\theta(N)$)

**การประกาศและการกำหนดค่าเริ่มต้นให้ vector** ใช้รูปแบบต่อไปนี้

- `vector<type> vec(N,a)` เพื่อสร้าง vector ขนาด `N` ที่บรรจุข้อมูล `a` ในทุกตำแหน่ง โดยที่ข้อมูล `a` ควรเป็นชนิด `T` (ถ้าไม่เป็น อาจเกิด compilation error หรือ implicit type conversion) (Time Complexity: $\theta(N)$)

- `vector<type> vec(vec1)` เพื่อสร้าง vector ใหม่ที่คัดลอกข้อมูลมาจาก vector เดิมที่ชื่อ `vec1` โดยที่ `vec` มีขนาดเท่ากับ `vec1` (Time Complexity: $\theta(\max(N,M))$ เมื่อ $N,M$ เป็นขนาดของ `vec` และ `vec1` ตามลำดับ )

**ตัวดำเนินการของ vector** (`N` คือขนาดของ vector)

| Operator | Description | Time Complexity |
|---|---|---|
| `Operator[i]` | Returns a reference to the element at position `i` in the vector. | $\theta(1)$ |
| `Operator=` | Assigns new content to the vector, replacing its current contents, resize to fit new content. | $O(\max(N,M))$ เมื่อ $N,M$ เป็นขนาดของ vector |

**ฟังก์ชันของ vector** (`N` คือขนาดของ vector `v`)

| Function | Description | Time Complexity |
|---|---|---|
| `v.size()` | Returns the number of elements in the vector. | $\theta(1)$ |
| `v.max_size()` | Returns the maximum number of elements that the vector can hold. | $\theta(1)$ |
| `v.resize(M)`<br>`v.resize(M, a)` | Resizes the vector so that it contains `M` elements.<br>Resizes the vector so that it contains `M` elements, if the new size is greater than the old one, fill all new positions with `a`. | $\theta(\lvert N-M\rvert)$ |
| `v.capacity()` | Returns the size of the storage space currently allocated for the vector (`vec.size() <= vec.capacity()`) | $\theta(1)$ |
| `v.empty()` | Returns `true` if the vector size is 0, `false` otherwise. | $\theta(1)$ |

| Function | Description | Time Complexity |
|---|---|---|
| `v.reserve(M)` | Requests that the vector capacity be at least enough to contain `M` elements. | $O(M)$ กรณี reallocation |
| `v.shrink_to_fit()` | Requests the vector to reduce its capacity to fit its size. | $O(N)$ |
| `v.at(i)` | Returns a reference to the element at position `i` in the vector. | $\theta(1)$ |
| `v.front()` | Returns a reference to the first element in the vector. | $\theta(1)$ |
| `v.back()` | Returns a reference to the last element in the vector | $\theta(1)$ |
| `v.data()` | Returns a direct pointer to the memory array used internally by the vector. | $\theta(1)$ |
| `v.assign(M, a)` | Assigns new contents of all `a` to the vector, replacing its current contents, resize to size `M`. | $O(\max(N,M))$ |
| `v.push_back(a)` | Adds a new element at the end of the vector. The content of `a` is copied (or moved) to the new element. | $\theta(1)$ หรือ $O(N)$ กรณี **reallocation** |
| `v.insert(it, a)`<br><br>`v.insert(it,n,a)` | The vector is extended by inserting a new element `a` before the element pointed by iterator `it`.<br>The vector is extended by inserting `n` new elements `a` before the element pointed by iterator `it`. | $O(n+N)$ |

| Function | Description | Time Complexity |
|---|---|---|
| `v.erase(it)`<br><br>`v.erase(f,l)` | Removes from the vector the element pointed by iterator `it`.<br>Removes from the vector elements pointed by iterator `f`, pointed by iterator `l`, and everything in between. | $O(n+N)$<br>เมื่อ `n` เป็นจำนวนข้อมูลที่ลบออก |
| `v.swap(vec)` | Exchanges the content of the vector by the content of `vec`, which is another vector object of the same type. | $\theta(1)$ |
| `v.clear()` | Removes all elements from the vector (which are destroyed) | $\theta(N)$ |
| `v.emplace(it, a)` | The vector is extended by inserting a new element `a` at position pointed by iterator `it`. This new element is constructed in place without allocating storage for the element. | $O(N)$ |
| `v.emplace_back(a)` | Inserts a new element `a` at the end of the vector. This new element is constructed in place without allocating storage for the element. | $\theta(1)$<br>หรือ<br>$O(N)$<br>กรณี **reallocation** |

**Iterator ของ vector**

| Function | Description | Time Complexity |
|---|---|---|
| `v.begin()` | Returns an iterator pointing to the first element in the vector. | $\theta(1)$ |
| `v.end()` | Returns an iterator pointing to the last element in the vector. | $\theta(1)$ |
| `v.rbegin()` | Return reverse iterator to reverse beginning. | $\theta(1)$ |
| `v.rend()` | Return reverse iterator to reverse ending. | $\theta(1)$ |
| `v.cbegin()` | Returns a const_iterator pointing to the first element in the vector. | $\theta(1)$ |
| `v.cend()` | Returns a const_iterator pointing to the last element in the vector | $\theta(1)$ |
| `v.crbegin()` | Return reverse const_iterator to reverse beginning. | $\theta(1)$ |
| `v.crend()` | Return reverse const_iterator to reverse ending. | $\theta(1)$ |

**** Time Complexity แสดงในกรณีที่ไม่มีการ reallocation ยกเว้นกำกับ**

**ข้อแตกต่างและการเลือกใช้ vector และ list**

- list ใช้หน่วยความจำในการเก็บข้อมูลมากกว่า vector สำหรับข้อมูลชนิดเดียวกัน จำนวนเท่ากัน เพราะต้องเก็บ pointer ด้วย
- list ไม่จำเป็นต้อง allocate หน่วยความจำสำหรับข้อมูลที่ยังไม่ได้นำมาเก็บ ในขณะที่ vector ต้อง allocate หน่วยความจำให้เพียงพอกับ capacity ของ vector แม้ว่าจะยังไม่เก็บข้อมูลใด ๆ
- การเข้าถึงข้อมูลหลายตัวตามลำดับที่เก็บ vector ทำได้เร็วกว่า list เพราะข้อมูลเก็บไว้ในหน่วยความจำที่อยู่ติดกัน และ vector มีโอกาสที่จะเก็บข้อมูลทั้งหมดไว้ใน cache ได้ ทำให้ CPU สามารถเข้าถึงข้อมูลได้เร็วที่สุด (เมื่อเทียบกับข้อมูลที่เก็บใน RAM)
- ข้อมูลใน vector สามารถเข้าถึงแบบ random access ได้เมื่อทราบตำแหน่ง/Index (Time Complexity: $\theta(1)$) ในขณะที่ list ไม่สามารถเข้าถึงข้อมูลแบบ random access ได้
- แม้ว่าการเพิ่มและลบข้อมูล 1 ตัวที่อยู่ตำแหน่งสุดท้ายของ vector มี time complexity $\theta(1)$ (ถ้าไม่ต้องทำ reallocation) แต่การเพิ่มและลบข้อมูล 1 ตัวที่ไม่อยู่ตำแหน่งสุดท้ายมี time complexity $O(N)$ ในขณะที่การเพิ่มและลบข้อมูล 1 ตัวที่ตำแหน่งใด ๆ ของ list มี time complexity $\theta(1)$
- เลือกใช้ vector ถ้าทราบจำนวนข้อมูลที่จะเก็บอย่างแน่นอน หรือ หากการเพิ่มหรือลบข้อมูลจะเกิดขึ้นกับข้อมูลตัวสุดท้ายเท่านั้น หรือ มีความจำเป็นที่ต้องเข้าถึงข้อมูลหลายตัวที่อยู่ติดกันบ่อยครั้ง
- เลือกใช้ list ถ้าต้องมีการเพิ่มหรือลบข้อมูลที่ตำแหน่งใด ๆ บ่อยครั้ง
- ในกรณีที่ต้องมีการเพิ่มหรือลบข้อมูลที่ตำแหน่งแรก และตำแหน่งสุดท้ายเท่านั้น สามารถเลือกใช้ STL container ที่ชื่อ deque ได้ (deque เป็น dynamic array เหมือน vector ยกเว้นแต่สามารถเพิ่ม (push_front) หรือลบ (pop_front) ข้อมูลที่ตำแหน่งแรกด้วย time complexity $\theta(1)$

**โปรแกรมที่ 1.1** การรับค่ามาเก็บใน vector, การปรับค่าใน vector, การเรียงลำดับค่าใน vector ด้วย STL sort( ) และการแสดงผลค่าใน vector (Time Complexity: $O(N \log N)$)

```
1.  #include<iostream>
2.  #include<vector>          // vector
3.  #include<algorithm>       // sort
4.  using namespace std;
5.
6.  // Forwards
7.  void printVector(const vector<int> &vec);
8.  void updateVectorA(vector<int> &vec);
9.  void updateVectorB(vector<int> &vec);
10.
11. int main(){
12.     int N; cin >> N;
13.     vector<int> data;    // input data
14.     data.resize(N);      // O(N)
15.     for(auto &i : data)  // O(N)
16.         cin>>i;
17.     updateVectorA(data);
18.     updateVectorB(data);
19.     cout <<"After update\n";
20.     printVector(data);
21.     cout <<"After sorting\n";
22.     sort(data.begin(),data.end()); // O(NlogN)
23.     printVector(data);
24.     cout << endl;
25.     return 0;
26. }
27. void printVector(const vector<int> &vec){
28.     for(auto &i : vec)  // O(N)
29.         cout<<i<< " ";
30.     cout << "\n";
31. }
32. void updateVectorA(vector<int> &vec){
33.     for(auto &i : vec) // O(N)
34.         i = i*2;
35. }
36. void updateVectorB(vector<int> &vec){
37.     for(auto it=vec.begin();it!=vec.end();it++)  // O(N)
38.         *it = (*it)*3;
39. }
```

**ผลลัพธ์**

```
6
6 5 4 3 2 1
After update
36 30 24 18 12 6
After sorting
6 12 18 24 30 36
```

**โปรแกรมที่ 1.2** การรับค่ามาเก็บใน vector, การปรับค่าใน vector, และการแสดงผลค่าใน vector โดยใช้ตัวดำเนินการ [ ] เช่นเดียวกับอาร์เรย์ (Time Complexity: $O(N)$)

```
1.  #include<iostream>
2.  #include<vector>          // vector
3.  using namespace std;
4.
5.  // Forwards
6.  void printVector(const vector<int> &vec, const int &N);
7.  void updateVector(vector<int> &vec, const int &N);
8.
9.  int main(){
10.     int N; cin >> N;
11.     vector<int> data;    // input data
12.     data.resize(N);      // O(N)
13.     for(int i; i<N; i++)  // O(N)
14.         cin >> data[i];
15.     updateVector(data, N);
16.     cout << "After update\n";
17.     printVector(data, N);
18.     cout << endl;
19.     return 0;
20. }
21. void updateVector(vector<int> &vec, const int &N){
22.     for(int i; i<N; i++)  // O(N)
23.         vec[i] *= 2;
24. }
25. void printVector(const vector<int> &vec, const int &N){
26.     for(int i; i<N; i++)  // O(N)
27.         cout<< vec[i] << " ";
28.     cout << "\n";
29. }
```

**โปรแกรมที่ 1.3** การใช้ vector ในการเก็บค่า และแสดงผลในรูป 2 มิติ (Time Complexity: $O(MN)$)

```
1.  #include<iostream>
2.  #include<vector>          // vector
3.  using namespace std;
4.
5.  // Forwards
6.  void print2DVectorA(const vector< vector <int> > &vec);
7.  void print2DVectorB(const vector< vector <int> > &vec);
8.  void print2DVectorC(const vector< vector <int> > &vec,
9.                      const int &M, const int &N);
10.
11. int main(){
12.     int M,N; cin >> M >> N;
13.     vector< vector<int> > data;  // input data
14.     data.resize(M);
15.     for(auto &r : data){     // O(MN)
16.         r.resize(N);
17.         for(auto &c : r)
18.             cin >> c;
19.     }
20.     cout << "Range-based loop\n";
21.     print2DVectorA(data);
22.     cout << "Iterator\n";
23.     print2DVectorB(data);
24.     cout << "Access with operator[ ]\n";
25.     print2DVectorC(data, M, N);
26.     return 0;
27. }
28. // Range-based loop method
29. void print2DVectorA(const vector< vector <int> > &vec){
30.     for(auto &r : vec){      // O(MN)
31.         for(auto &c : r)
32.             cout << c << " ";
33.         cout << "\n";
34.     }
35.     cout << "\n";
36. }
37. // Iterator method
38. void print2DVectorB(const vector< vector <int> > &vec){
39.     for(auto rit=vec.begin();rit!=vec.end();rit++){  // O(MN)
40.         for(auto cit=rit->begin();cit!=rit->end();cit++)
41.             cout << *cit << " ";
42.         cout << "\n";
43.     }
44.     cout << "\n";
45. }
46.
47.
```

```
48.  // Access with operator[ ]
49.  void print2DVectorC(const vector< vector <int> > &vec,
50.                        const int &M, const int &N){
51.      for(int r=0; r<M; r++){        // O(MN)
52.          for(int c=0; c<N; c++)
53.              cout << vec[r][c] << " ";
54.          cout << "\n";
55.      }
     }
```

**ผลลัพธ์**

**3 4**

**1 2 3 4**

**5 6 7 8**

**9 1 2 3**

```
Range-based loop
1 2 3 4
5 6 7 8
9 1 2 3

Iterator
1 2 3 4
5 6 7 8
9 1 2 3

Access with operator[ ]
1 2 3 4
5 6 7 8
9 1 2 3
```